Automata with Generalized Rabin Pairs for Probabilistic Model Checking and LTL Synthesis

Krishnendu Chatterjee[1⋆], Andreas Gaiser[2⋆⋆], and Jan Křetínský[2,3⋆⋆⋆]
[1] IST Austria
[2] Fakultät für Informatik, Technische Universität München, Germany
[3] Faculty of Informatics, Masaryk University, Brno, Czech Republic

Abstract. The model-checking problem for probabilistic systems crucially relies on the translation of LTL to deterministic Rabin automata (DRW). Our recent Safraless translation [KE12,GKE12] for the LTL(**F**,**G**) fragment produces smaller automata as compared to the traditional approach. In this work, instead of DRW we consider deterministic automata with acceptance condition given as disjunction of generalized Rabin pairs (DGRW). The Safraless translation of LTL(**F**,**G**) formulas to DGRW results in smaller automata as compared to DRW. We present algorithms for probabilistic model-checking as well as game solving for DGRW conditions. Our new algorithms lead to improvement both in terms of theoretical bounds as well as practical evaluation. We compare PRISM with and without our new translation, and show that the new translation leads to significant improvements.

1 Introduction

Logic for ω-regular properties. The class of ω-regular languages generalizes regular languages to infinite strings and provides a robust specification language to express all properties used in verification and synthesis. The most convenient way to describe specifications is through logic, as logics provide a concise and intuitive formalism to express properties with very precise semantics. The linear-time temporal logic (LTL) [Pnu77] is the de-facto logic to express linear time ω-regular properties in verification and synthesis.

Deterministic ω-automata. For model-checking purposes, LTL formulas can be converted to nondeterministic Büchi automata (NBW) [VW86], and then the problem reduces to checking emptiness of the intersection of two NBWs (representing the system and the negation of the specification, respectively). However, for two very important problems deterministic automata are used, namely, (1) the synthesis problem [Chu62,PR89]; and (2) the model-checking problem for probabilistic systems or Markov decision processes (MDPs) [BK08] which has a wide range of applications from randomized communication, to security protocols, to biological systems. The standard approach is to translate LTL to NBW [VW86], and then convert the NBW to a deterministic automata with Rabin acceptance condition (DRW) using Safra's determinization procedure [Saf88] (or using a recent improvement of Piterman [Pit06]).

⋆ The author is supported by Austrian Science Fund (FWF) Grant No P 23499-N23, FWF NFN Grant No S11407-N23 (RiSE), ERC Start grant (279307: Graph Games), and Microsoft faculty fellows award.
⋆⋆ The author is supported by the DFG Graduiertenkolleg 1480 (PUMA).
⋆⋆⋆ The author is supported by the Czech Science Foundation, project No. P202/10/1469

Avoiding Safra's construction. The key bottleneck of the standard approach in practice is Safra's determinization procedure which is difficult to implement due to the complicated state space and data structures associated with the construction [Kup12]. As a consequence several alternative approaches have been proposed, and the most prominent ones are as follows. The first approach is the Safraless approach. One can reduce the synthesis problem to emptiness of nondeterministic Büchi tree automata [KV05]; it has been implemented with considerable success in [JB06]. For probabilistic model checking other constructions can be also used, however, all of them are exponential [**?**,**?**]. The second approach is to use heuristic to improve Safra's determinization procedure [KB06,KB07] which has led to the tool ltl2dstar [Kle]. The third approach is to consider fragments of LTL. In [AT04] several simple fragments of LTL were proposed that allow much simpler (single exponential as compared to the general double exponential) translations to deterministic automata. The generalized reactivity(1) fragment of LTL (called GR(1)) was introduced in [PPS06] and a cubic time symbolic representation of an equivalent automaton was presented. The approach has been implemented in the ANZU tool [JGWB07]. Recently, the $(\mathbf{F}, \mathbf{G})$-fragment of LTL, that uses boolean operations and only $\mathbf{F}$ (eventually or in future) and $\mathbf{G}$ (always or globally) as temporal operators, was considered and a simple and direct translation to deterministic Rabin automata (DRW) was presented [KE12]. Not only it covers all fragments of [AT04], but it can also express all complex fairness constraints, which are widely used in verification.

Probabilistic model-checking. Despite several approaches to avoid Safra's determinization, for probabilistic model-checking the deterministic automata are still necessary. Since probabilistic model-checkers handle linear arithmetic, they do not benefit from the symbolic methods of [PPS06,MS08] or from the tree automata approach. The approach for probabilistic model-checking has been to explicitly construct a DRW from the LTL formula. The most prominent probabilistic model-checker PRISM [KNP11] implements the ltl2dstar approach.

Our results. In this work, we focus on the $(\mathbf{F}, \mathbf{G})$-fragment of LTL. Instead of the traditional approach of translation to DRW we propose a translation to deterministic automata with *generalized Rabin pairs*. We present probabilistic model-checking as well as symbolic game solving algorithms for the new class of conditions which lead to both theoretical as well as significant practical improvements. The details of our contributions are as follows.

1. A Rabin pair consists of the conjunction of a Büchi (always eventually) and a coBüchi (eventually always) condition, and a Rabin condition is a disjunction of Rabin pairs. A generalized Rabin pair is the conjunction of conjunctions of Büchi conditions and conjunctions of coBüchi conditions. However, as conjunctions of coBüchi conditions is again a coBüchi condition, a generalized Rabin pair is the conjunction of a coBüchi condition and conjunction of Büchi conditions.[†] We consider deterministic automata where the acceptance condition is a disjunction of generalized Rabin pairs (and call them

[†] Note that our condition (disjunction of generalized Rabin pairs) is very different from both generalized Rabin conditions (conjunction of Rabin conditions) and the

DGRW). The $(\mathbf{F}, \mathbf{G})$-fragment of LTL admits a direct and algorithmically simple translation to DGRW [KE12] and we consider DGRW for probabilistic model-checking and synthesis. The direct translation of LTL($\mathbf{F}$,$\mathbf{G}$) could be done to a compact deterministic automaton with a Muller condition, however, the explicit representation of the Muller condition is typically huge and not algorithmically efficient, and thus reduction to deterministic Rabin automata was performed (with a blow-up) since Rabin conditions admit efficient algorithmic analysis. We show that DGRW allow both for a very compact translation of the $(\mathbf{F}, \mathbf{G})$-fragment of LTL as well as efficient algorithmic analysis. The direct translation of LTL($\mathbf{F}$,$\mathbf{G}$) to DGRW has the same number of states as for a general Muller condition. For many formulae expressing e.g. fairness-like conditions the translation to DGRW is significantly more compact than the previous ltl2dstar approach. For example, for a conjunction of three strong fairness constraints, ltl2dstar produces a DRW with more than a million states, translation to DRW via DGRW requires 469 states, and the corresponding DGRW has only 64 states.

2. One approach for probabilistic model-checking and synthesis for DGRW would be to first convert them to DRW, and then use the standard algorithms. Instead we present direct algorithms for DGRW that avoids the translation to DRW both for probabilistic model-checking and game solving. The direct algorithms lead to both theoretical and practical improvements. For example, consider the disjunctions of k generalized Rabin pairs such that in each pair there is a conjunction of a coBüchi condition and conjunctions of j Büchi conditions. Our direct algorithms for probabilistic model-checking as well as game solving is more efficient by a multiplicative factor of j^k and j^{k^2+k} as compared to the approach of translation to DRW for probabilistic model checking and game solving, respectively. Moreover, we also present symbolic algorithms for game solving for DGRW conditions.
3. We have implemented our approach for probabilistic model checking in PRISM, and the experimental results show that as compared to the existing implementation of PRISM with ltl2dstar our approach results in improvement of order of magnitude. Moreover, the results for games confirm that the speed up is even greater than for probabilistic model checking.

2 Preliminaries

In this section, we recall the notion of linear temporal logic (LTL) and illustrate the recent translation of its ($\mathbf{F}$,$\mathbf{G}$)-fragment to DRW [KE12,GKE12] through the intermediate formalism of DGRW. Finally, we define an index that is important for characterizing the savings the new formalism of DGRW brings as shown in the subsequent sections.

generalized Rabin(1) condition of [Ehl11], which considers a set of assumptions and guarantees where each assumption and guarantee consists of *one* Rabin pair. Syntactically, disjunction of generalized Rabin pairs condition is $\bigvee_i(\mathbf{FG}a_i \wedge \bigwedge_j \mathbf{GF}b_{ij})$, whereas generalized Rabin condition is $\bigwedge_j(\bigvee_i(\mathbf{FG}a_{ij} \wedge \mathbf{GF}b_{ij}))$, and generalized Rabin(1) condition is $(\bigwedge_i(\mathbf{FG}a_i \wedge \mathbf{GF}b_i) \Rightarrow \bigwedge_j(\mathbf{FG}a_j \wedge \mathbf{GF}b_j))$.

2.1 Linear temporal logic

We start by recalling the fragment of linear temporal logic with *future* (**F**) and *globally* (**G**) modalities.

Definition 1 (LTL(F,G) syntax). *The formulae of the* (**F**,**G**)-fragment of linear temporal logic *are given by the following syntax:*

$$\varphi ::= a \mid \neg a \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathbf{F}\varphi \mid \mathbf{G}\varphi$$

where a ranges over a finite fixed set Ap of atomic propositions.

We use the standard abbreviations $\mathbf{tt} := a \vee \neg a$ and $\mathbf{ff} := a \wedge \neg a$. Note that we use the negation normal form, as negations can be pushed inside to atomic propositions due to the equivalence of $\mathbf{F}\varphi$ and $\neg\mathbf{G}\neg\varphi$.

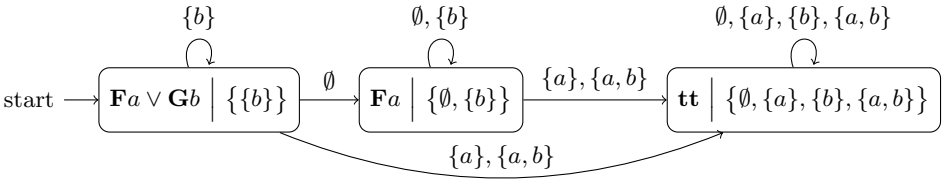
Definition 2 (LTL(F,G) semantics). *Let $w \in (2^{Ap})^\omega$ be a word. The ith letter of w is denoted $w[i]$, i.e. $w = w[0]w[1]\cdots$. Further, we define the ith suffix of w as $w_i = w[i]w[i+1]\cdots$. The semantics of a formula on w is then defined inductively as follows: $w \models a \iff a \in w[0]$; $w \models \neg a \iff a \notin w[0]$; $w \models \varphi \wedge \psi \iff w \models \varphi$ and $w \models \psi$; $w \models \varphi \vee \psi \iff w \models \varphi$ or $w \models \psi$; and*

$$\begin{aligned} w \models \mathbf{F}\varphi &\iff \exists\, k \in \mathbb{N}_0 : w_k \models \varphi \\ w \models \mathbf{G}\varphi &\iff \forall\, k \in \mathbb{N}_0 : w_k \models \varphi \end{aligned}$$

2.2 Translating LTL(F,G) into deterministic ω-automata

Recently, in [KE12,GKE12], a new translation of LTL(**F**,**G**) to deterministic automata has been proposed. This construction avoids Safra's determinization and makes direct use of the structure of the formula. We illustrate the construction in the following examples.

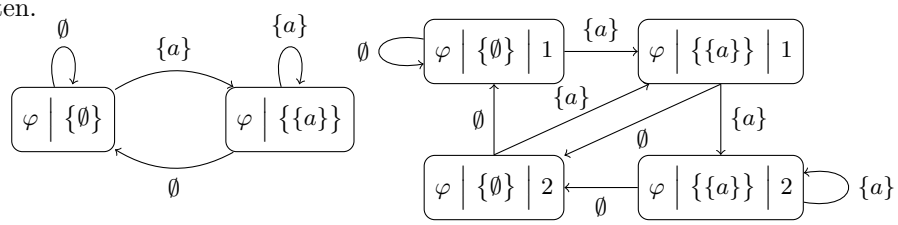
Example 3. Consider a formula $\mathbf{F}a \vee \mathbf{G}b$. The construction results in the following automaton. The state space of the automaton has two components. The first component stores the current formula to be satisfied. Whenever a letter is read, the formula is updated accordingly. For example, when reading a letter with no b, the option to satisfy the formula due to satisfaction of $\mathbf{G}b$ is lost and is thus reflected in changing the current formula to $\mathbf{F}a$ only.



The second component stores the last letter read (actually, an equivalence class thereof). The purpose of this component is explained in the next example. For formulae with no mutual nesting of **F** and **G** this component is redundant.

The formula $\mathbf{F}a \vee \mathbf{G}b$ is satisfied either due to $\mathbf{F}a$ or $\mathbf{G}b$. Therefore, when viewed as a Rabin automaton, there are two Rabin pairs. One forcing infinitely many visits of the third state (a in $\mathbf{F}a$ must be eventually satisfied) and the other prohibiting infinitely many visits of the second and third states (b in $\mathbf{G}b$ must never be violated). The acceptance condition is a disjunction of these pairs.

Example 4. Consider now the formula $\varphi = \mathbf{GF}a \wedge \mathbf{GF}\neg a$. Satisfaction of this formula does not depend on any finite prefix of the word and reading $\{a\}$ or $\emptyset$ does not change the first component of the state. This infinitary behaviour requires the state space to record which letters have been seen infinitely often and the acceptance condition to deal with that. In this case, satisfaction requires visiting the second state infinitely often *and* visiting the first state infinitely often.



However, such a conjunction cannot be written as a Rabin condition. In order to get a Rabin automaton, we would duplicate the state space. In the first copy, we wait for reading $\{a\}$. Once this happens we move to the second copy, where we wait for reading $\emptyset$. Once we succeed we move back to the first copy and start again. This bigger automaton now allows for a Rabin condition. Indeed, it is sufficient to infinitely often visit the "successful" state of the last copy as this forces infinite visits of "successful" states of all copies.

In order to obtain a DRW from an LTL formula, [KE12,GKE12] first constructs an automaton similar to DGRW (like the one on the left) and then the state space is blown-up and a DRW (like the one on the right) is obtained. However, we shall argue that this blow-up is unnecessary for application in probabilistic model checking and in synthesis. This will result in much more efficient algorithms for complex formulae. In order to avoid the blow-up we define and use DGRW, an automaton with more complex acceptance condition, yet as we show algorithmically easy to work with and efficient as opposed to e.g. the general Muller condition.

2.3 Automata with generalized Rabin pairs

In the previous example, the cause of the blow-up was the conjunction of Rabin conditions. In [KE12], a generalized version of Rabin condition is defined that allows for capturing conjunction. It is defined as a positive Boolean combination of Rabin pairs. Whether a set $\mathrm{Inf}(\rho)$ of states visited infinitely often on a run ρ is accepting or not is then defined inductively as follows:

$$\begin{aligned}
\mathrm{Inf}(\rho) \models \varphi \wedge \psi &\iff \mathrm{Inf}(\rho) \models \varphi \text{ and } \mathrm{Inf}(\rho) \models \psi \\
\mathrm{Inf}(\rho) \models \varphi \vee \psi &\iff \mathrm{Inf}(\rho) \models \varphi \text{ or } \mathrm{Inf}(\rho) \models \psi \\
\mathrm{Inf}(\rho) \models (F, I) &\iff F \cap \mathrm{Inf}(\rho) = \emptyset \text{ and } I \cap \mathrm{Inf}(\rho) \neq \emptyset
\end{aligned}$$

Denoting Q as the set of all states, (F, I) is then equivalent to $(F, Q) \wedge (\emptyset, I)$. Further, $(F_1, Q) \wedge (F_2, Q)$ is equivalent to $(F_1 \cup F_2, Q)$. Therefore, one can transform any such condition into a disjunctive normal form and obtain a condition of the following form:

$$\bigvee_{i=1}^{k} \left(\left(F_i, Q\right) \wedge \bigwedge_{j=1}^{\ell_i} \left(\emptyset, I_i^j\right) \right) \qquad (*)$$

Therefore, in this paper we define the following new class of ω-automata:

Definition 5 (DGRW). *An* automaton with generalized Rabin pairs (DGRW) *is a (deterministic) ω-automaton $\mathcal{A} = (Q, q_0, \delta)$ over an alphabet Σ, where Q is a set of states, q_0 is the initial state, $\delta : Q \times \Sigma \to Q$ is a transition function, together with a* generalized Rabin pairs (GRP) acceptance condition $\mathcal{GR} \subseteq 2^{2^Q \times 2^{2^Q}}$. *A run ρ of $\mathcal{A}$ is accepting for $\mathcal{GR} = \left\{ \left(F_i, \{I_i^1, \ldots, I_i^{\ell_i}\}\right) \;\middle|\; i \in \{1, \ldots, k\} \right\}$ if there is $i \in \{1, \ldots, k\}$ such that*

$$F_i \cap \mathrm{Inf}(\rho) = \emptyset \text{ and}$$
$$I_i^j \cap \mathrm{Inf}(\rho) \neq \emptyset \text{ for every } j \in \{1, \ldots, \ell_i\}$$

Each $(F_i, \mathcal{I}_i) = \left(F_i, \{I_i^1, \ldots, I_i^{\ell_i}\}\right)$ is called a generalized Rabin pair (GRP)*, and the* GRP condition *is thus a disjunction of generalized Rabin pairs..*

W.l.o.g. we assume $k > 0$ and $\ell_i > 0$ for each $i \in \{1, \ldots, k\}$ (whenever $\ell_i = 0$ we could set $\mathcal{I}_i = \{Q\}$). Although the type of the condition allows for huge instances of the condition, the construction of [KE12] (producing this disjunctive normal form) guarantees efficiency not worse than that of the traditional determinization approach. For a formula of size n, it is guaranteed that $k \leq 2^n$ and $\ell_i \leq n$ for each $i \in \{1, \ldots, k\}$. Further, the size of the state space is at most $2^{\mathcal{O}(2^n)}$. Moreover, consider *"infinitary"* formulae, where each atomic proposition has both **F** and **G** as ancestors in the syntactic tree of the formula. Since the first component of the state space is always the same, the size of the state space is bounded by $2^{|Ap|}$ as the automaton only remembers the last letter read. We will make use of this fact later.

2.4 Degeneralization

As already discussed, one can blow up any automaton with generalized Rabin pairs and obtain a Rabin automaton. We need the following notation. For any $n \in \mathbb{N}$, let $[1..n]$ denote the set $\{1, \ldots, n\}$ equipped with the operation $\oplus$ of cyclic addition, i.e. $m \oplus 1 = m + 1$ for $m < n$ and $n \oplus 1 = 1$.

The DGRW defined above can now be degeneralized as follows. For each $i \in \{1, \ldots, k\}$, multiply the state space by $[1..\ell_i]$ to keep track for which I_i^j we are currently waiting for. Further, adjust the transition function so that we leave the jth copy once we visit I_i^j and immediately go to the next copy. Formally, for $\sigma \in \Sigma$ set $(q, w_1, \ldots, w_k) \xrightarrow{\sigma} (r, w_1', \ldots, w_k')$ if $q \xrightarrow{\sigma} r$ and $w_i' = w_i$ for all i with $q \notin I_i^{w_i}$ and $w_i' = w_i \oplus 1$ otherwise.

The resulting blow-up factor is then the following:

Definition 6 (Degeneralization index). *For a GRP condition* $\mathcal{GR} = \{(F_i, \mathcal{I}_i) \mid i \in [1..k]\}$*, we define the* degeneralization domain $B := \prod_{i=1}^{k} [1..|\mathcal{I}_i|]$ *and the* degeneralization index of $\mathcal{GR}$ *to be* $|B| = \prod_{i=1}^{k} |\mathcal{I}_i|$.

The state space of the resulting Rabin automaton is thus $|B|$-times bigger and the number of pairs stays the same. Indeed, for each $i \in \{1, \ldots, k\}$ we have a Rabin pair

$$\left(F_i \times B, I_i^{\ell_i} \times \{b \in B \mid b(i) = \ell_i\}\right)$$

Example 7. In Example 3 there is one pair and the degeneralization index is 2.

Example 8. For a conjunction of three fairness constraints $\varphi = (\mathbf{FG}a \vee \mathbf{GF}b) \wedge (\mathbf{FG}c \vee \mathbf{GF}d) \wedge (\mathbf{FG}e \vee \mathbf{GF}f)$, the Büchi components $\mathcal{I}_i$'s of the equivalent GRP condition correspond to $\mathbf{tt}, b, d, f, b \wedge d, b \wedge f, d \wedge f, b \wedge d \wedge f$. The degeneralization index is thus $|B| = 1 \cdot 1 \cdot 1 \cdot 1 \cdot 2 \cdot 2 \cdot 2 \cdot 3 = 24$. For four constraints, it is $1 \cdot 1^4 \cdot 2^6 \cdot 3^4 \cdot 4 = 20736$. One can easily see the index grows doubly exponentially.

3 Probabilistic Model Checking

In this section, we show how automata with generalized Rabin pairs can significantly speed up model checking of Markov decision processes (i.e., probabilistic model checking). For example, for the fairness constraints of the type mentioned in Example 8 the speed-up is by a factor that is doubly exponential. Although there are specialized algorithms for checking properties under strong fairness constraints (implemented in PRISM), our approach is general and speeds up for a wide class of constraints. The combinations (conjunctions, disjunctions) of properties not expressible by small Rabin automata (and/or Streett automata) are infeasible for the traditional approach, while we show that automata with generalized Rabin pairs often allow for efficient model checking. First, we present the theoretical model-checking algorithm for the new type of automata and the theoretical bounds for savings. Second, we illustrate the effectiveness of the approach experimentally.

3.1 Model checking using generalized Rabin pairs

We start with the definitions of Markov decision processes (MDPs), and present the model-checking algorithms. For a finite set V, let $\mathrm{Distr}(V)$ denote the set of probability distributions on V.

Definition 9 (MDP and MEC). *A* Markov decision process (MDP) $\mathcal{M} = (V, E, (V_0, V_P), \delta)$ *consists of a finite directed* MDP graph (V, E)*, a partition* (V_0, V_P) *of the* finite *set* V *of vertices into player-0 vertices* (V_0) *and probabilistic vertices* (V_P)*, and a probabilistic transition function* δ*:* $V_P \to \mathrm{Distr}(V)$ *such that for all vertices* $u \in V_P$ *and* $v \in V$ *we have* $(u, v) \in E$ *iff* $\delta(u)(v) > 0$.

An end-component U *of an MDP is a set of its vertices such that (i) the subgraph induced by* U *is strongly connected and (ii) for each edge* $(u, v) \in E$*, if* $u \in U \cap V_P$*, then* $v \in U$ *(i.e., no probabilistic edge leaves* U*).*

A maximal end-component (MEC) *is an end-component that is maximal w.r.t. to the inclusion ordering.*

If U_1 and U_2 are two end-components and $U_1 \cap U_2 \neq \emptyset$, then $U_1 \cup U_2$ is also an end-component. Therefore, every MDP induces a unique set of its MECs, called *MEC decomposition*.

For precise definition of semantics of MDPs we refer to [Put94]. Note that MDPs are also defined in an equivalent way in literature with a set of actions such that every vertex and choice of action determines the probability distribution over the successor states; the choice of actions corresponds to the choice of edges at player-0 vertices of our definition.

The standard model-checking algorithm for MDPs proceeds in several steps. Given an MDP $\mathcal{M}$ and an LTL formula φ

1. compute a deterministic automaton $\mathcal{A}$ recognizing the language of φ,
2. compute the product $\overline{\mathcal{M}} = \mathcal{M} \times \mathcal{A}$,
3. solve the product MDP $\overline{\mathcal{M}}$.

The algorithm is generic for all types of deterministic ω-automata $\mathcal{A}$. The leading probabilistic model checker PRISM [KNP11] re-implements ltl2dstar [Kle] that transforms φ into a deterministic *Rabin* automaton. This approach employs Safra's determinization and thus despite many optimization often results in an unnecessarily big automaton.

There are two ways to fight the problem. Firstly, one can strive for smaller Rabin automata. Secondly, one can employ other types of ω-automata. As to the former, we have plugged our implementation Rabinizer [GKE12] of the approach [KE12] into PRISM, which already results in considerable improvement. For the latter, Example 4 shows that Muller automata can be smaller than Rabin automata. However, explicit representation of Muller acceptance conditions is typically huge. Hence the third step to solve the product MDP would be too expensive. Therefore, we propose to use automata with generalized Rabin pairs.

On the one hand, DGRW often have small state space after translation. Actually, it is the same as the state space of the intermediate Muller automaton of [KE12]. Compared to the corresponding naively degeneralized DRW it is $|B|$ times smaller (one can still perform some optimizations in the degeneralization process, see the experimental results).

On the other hand, as we show below the acceptance condition is still algorithmically efficient to handle. We now present the steps to solve the product MDP for a GRP acceptance condition, i.e. a disjunction of generalized Rabin pairs. Consider an MDP with k generalized Rabin pairs $(F_i, \{I_i^1, \ldots, I_i^{\ell_i}\})$, for $i = 1, 2, \ldots, k$. The steps of the computation are as follows:

1. For $i = 1, 2, \ldots, k$;
   (a) Remove the set of states F_i from the MDP.
   (b) Compute the MEC decomposition.
   (c) If a MEC C has a non-empty intersection with each I_i^j, for $j = 1, 2, \ldots, \ell_i$, then include C as a winning MEC.
   (d) let W_i be the union of winning MECs (for the ith pair).
2. Let W be the union of W_i, i.e. $W = \bigcup_{i=1}^{k} W_i$.
3. The solution (or optimal value of the product MDP) is the maximal probability to reach the set W.

Given an MDP with n vertices and m edges, let $\mathsf{MEC}(n,m)$ denote the complexity of computing the MEC decomposition; and $\mathsf{LP}(n,m)$ denotes the complexity to solve linear-programming solution with m constraints over n variables.

Theorem 10. *Given an MDP with n vertices and m edges with k generalized Rabin pairs $(F_i, \{I_i^1, \ldots, I_i^{\ell_i}\})$, for $i = 1, 2, \ldots, k$, the solution can be achieved in time $\mathcal{O}(k \cdot \mathsf{MEC}(n,m) + n \cdot \sum_{i=1}^{k} \ell_i) + \mathcal{O}(\mathsf{LP}(n,m))$.*

Remark 11. The best known complexity to solve MDPs with Rabin conditions of k pairs require time $\mathcal{O}(k \cdot \mathsf{MEC}(n,m)) + \mathcal{O}(\mathsf{LP}(n,m))$ time [**?**]. Thus degeneralization of generalized Rabin pairs to Rabin conditions and solving MDPs would require time $\mathcal{O}(k \cdot \mathsf{MEC}(|B| \cdot n, |B| \cdot m)) + \mathcal{O}(\mathsf{LP}(|B| \cdot n, |B| \cdot m))$ time. The current best known algorithms for maximal end-component decomposition require at least $\mathcal{O}(m \cdot n^{2/3})$ time [**?**], and the simplest algorithms that are typically implemented require $\mathcal{O}(n \cdot m)$ time. Thus our approach is more efficient at least by a factor of $B^{5/3}$ (given the current best known algorithms), and even if both maximal end-component decomposition and linear-programming can be solved in linear time, our approach leads to a speed-up by a factor of $|B|$, i.e. exponential in $\mathcal{O}(k)$ the number of non-trivially generalized Rabin pairs. In general if $\beta \geq 1$ is the sum of the exponents required to solve the MEC decomposition (resp. linear-programming), then our approach is better by a factor of $|B|^{\beta}$.

Example 12. A Rabin automaton for n constraints of Example 8 is of doubly exponential size, which is also the factor by which the product and thus the running time grows. However, as the formula is "infinitary" (see end of Section 2.3), the state space of the generalized automaton is 2^{Ap} and the product is of the very same size as the original system since the automaton only monitors the current labelling of the state.

3.2 Experimental results

In this section, we compare the performance of

- **Ł** the original PRISM with its implementation of ltl2dstar producing Rabin automata,
- **R** PRISM with Rabinizer [GKE12] (our implementation of [KE12]) producing DRW via *optimized* degeneralization of DGRW, and
- **GR** PRISM with Rabinizer producing DGRW and with the modified MEC checking step.

We have performed a case study on the Pnueli-Zuck randomized mutual exclusion protocol [PZ86] implemented as a PRISM benchmark. We consider the protocol with 3, 4, and 5 participants. The sizes of the respective models are $s_3 = 2\,368$, $s_4 = 27\,600$, and $s_5 = 308\,800$ states. We have checked these models against several formulae illustrating the effect of the degeneralization index on the speed up of our method; see Table 1.

In the first column, there are the formulae in the form of a PRISM query. We ask for a maximal/minimal value over all schedulers. Therefore, in the P_{max} case, we create an automaton for the formula, whereas in the case of P_{min} we

create an automaton for its negation. The second column then states the number i of participants, thus inducing the respective size s_i of the model.

The next three columns depict the size of the product of the system and the automaton, for each of the **Ł**, **R**, **GR** variants. The size is given as the ratio of the actual size and the respective s_i. The number then describes also the "effective" size of the automaton when taking the product. The next three columns display the total running times for model checking in each variant.

The last three columns illustrate the efficiency of our approach. The first column $t_{\mathbf{R}}/t_{\mathbf{GR}}$ states the time speed-up of the DGRW approach when compared to the corresponding degeneralization. The second column states the degeneralization index $|B|$. The last column $t_{\mathbf{Ł}}/t_{\mathbf{GR}}$ then displays the overall speed-up of our approach to the original PRISM.

In the formulae, an atomic proposition $p_i = j$ denotes that the ith participant is in its state j. The processes start in state 0. In state 1 they want to enter the critical section. State 10 stands for being in the critical section. After leaving the critical section, the process re-enters state 0 again.

Table 1. Experimental comparison of **Ł**, **R**, and **GR** methods. All measurements performed on Intel i7 with 8 GB RAM. The sign "–" denotes either crash, out-of-memory, time-out after 30 minutes, or a ratio where one operand is –.

Formula	#	$\frac{s_{\mathbf{Ł}}}{s_i}$	$\frac{s_{\mathbf{R}}}{s_i}$	$\frac{s_{\mathbf{GR}}}{s_i}$	$t_{\mathbf{Ł}}$	$t_{\mathbf{R}}$	$t_{\mathbf{GR}}$	$\frac{t_{\mathbf{R}}}{t_{\mathbf{GR}}}$	$\lvert B\rvert$	$\frac{t_{\mathbf{Ł}}}{t_{\mathbf{GR}}}$
P_{max} =?[$\mathbf{GF}p_1{=}10$	3	4.1	2.6	1	1.2	0.4	0.2	2.2	3	6.8
$\wedge\ \mathbf{GF}p_2{=}10$	4	4.3	2.7	1	17.4	1.8	0.3	6.4	3	60.8
$\wedge\ \mathbf{GF}p_3{=}10$]	5	4.4	2.7	1	257.5	15.2	0.6	26.7	3	447.9
P_{max} =?[$\mathbf{GF}p_1{=}10 \wedge \mathbf{GF}p_2{=}10$	4	6	3.5	1	27.3	2.5	0.9	2.8	4	32.1
$\wedge\ \mathbf{GF}p_3{=}10 \wedge \mathbf{GF}p_4{=}10$]	5	6.2	3.6	1	408.5	17.8	0.9	20.4	4	471.2
P_{min} =?[$\mathbf{GF}p_1{=}10 \wedge \mathbf{GF}p_2{=}10$	4	–	1	1	–	36.5	36.3	1	1	–
$\wedge\ \mathbf{GF}p_3{=}10 \wedge \mathbf{GF}p_4{=}10$]	5	–	1	1	–	610.6	607.2	1	1	–
P_{max} =?[($\mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_2{\neq}0$)	3	79.7	1.9	1	225.5	4.1	2.2	1.8	2	101.8
$\wedge(\mathbf{GF}p_2{=}0 \vee \mathbf{FG}p_3{\neq}0)$]	4	–	1.9	1	–	61.7	29.2	2.1	2	–
	5	–	1.9	1	–	1007	479	2.1	2	–
P_{max} =?[($\mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_1{\neq}0$)	3	23.3	1.9	1	66.4	3.92	2.2	1.8	2	30.7
$\wedge(\mathbf{GF}p_2{=}0 \vee \mathbf{FG}p_2{\neq}0)$]	4	23.3	1.9	1	551.5	61	28.2	2.2	2	19.6
	5	–	1.9	1	–	1002.7	463	2.2	2	–
P_{max} =?[($\mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_2{\neq}0$)	3	–	16.3	1	–	122.1	7.1	17.2	24	–
$\wedge(\mathbf{GF}p_2{=}0 \vee \mathbf{FG}p_3{\neq}0)$	4	–	–	1	–	–	75.6	–	24	–
$\wedge(\mathbf{GF}p_3{=}0 \vee \mathbf{FG}p_1{\neq}0)$]	5	–	–	1	–	–	1219.5	–	24	–
P_{max} =?[($\mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_1{\neq}0$)	3	–	12	1	–	76.3	7.2	12	24	–
$\wedge(\mathbf{GF}p_2{=}0 \vee \mathbf{FG}p_2{\neq}0)$	4	–	12.1	1	–	1335.6	78.9	19.6	24	–
$\wedge(\mathbf{GF}p_3{=}0 \vee \mathbf{FG}p_3{\neq}0)$]	5	–	–	1	–	–	1267.6	–	24	–
P_{min} =?[($\mathbf{GF}p_1{\neq}10 \vee \mathbf{GF}p_1{=}0 \vee \mathbf{FG}p_1{=}1$)	3	2.1	1	1	1.2	0.9	0.8	1	1	1.5
$\wedge\mathbf{GF}p_1{\neq}0 \wedge \mathbf{GF}p_1{=}1$]	4	2.1	1	1	11.8	8.7	8.8	1	1	1.3
	5	2.1	1	1	186.3	147.5	146.2	1	1	1.3
P_{max} =?[($\mathbf{G}p_1{\neq}10 \vee \mathbf{G}p_2{\neq}10 \vee \mathbf{G}p_3{\neq}10$)	3	–	32	5.9	–	405	80.1	5.1	8	–
$\wedge$ ($\mathbf{FG}p_1{\neq}1 \vee \mathbf{GF}p_2 = 1 \vee \mathbf{GF}p_3 = 1$)	4	–	–	6.4	–	–	703.5	–	8	–
$\wedge$ ($\mathbf{FG}p_2{\neq}1 \vee \mathbf{GF}p_1 = 1 \vee \mathbf{GF}p_3 = 1$)	5	–	–	–	–	–	–	–	8	–
P_{min} =?[($\mathbf{FG}p_1{\neq}0 \vee \mathbf{FG}p_2{\neq}0 \vee \mathbf{GF}p_3{=}0$)	3	55.9	4.7	1	289.7	12.6	3.4	3.7	12	84.3
$\vee$ ($\mathbf{FG}p_1{\neq}10 \wedge \mathbf{GF}p_2 = 10 \wedge \mathbf{GF}p_3 = 10$)	4	–	4.6	1	–	194.5	33.2	5.9	12	–
	5	–	–	1	–	–	543	–	12	–

Formulae 1 to 3 illustrate the effect of $|B|$ on the ratio of sizes of the product in the **R** and **GR** cases, see $\frac{s_{\mathbf{R}}}{s_i}$, and ratio of the required times. The theoretical prediction is that $s_{\mathbf{R}}/s_{\mathbf{GR}} = |B|$. Nevertheless, due to optimizations done in the degeneralization process, the first is often slightly smaller than the second one, see columns $\frac{s_{\mathbf{R}}}{s_i}$ and B. (Note that $s_{\mathbf{GR}}/s_i$ is 1 for "infinitary" formulae.) For the same reason, $\frac{t_{\mathbf{R}}}{t_{\mathbf{GR}}}$ is often smaller than $|B|$. However, with the growing size of the systems it gets bigger hence the saving factor is larger for larger systems, as discussed in the previous section.

Formulae 4 to 7 illustrate the doubly exponential growth of $|B|$ and its impact on systems of different sizes. The DGRW approach (**GR** method) is often the only way to create the product at all.

Formula 8 is a Streett condition showing the approach still performs competitively. Formulae 9 and 10 combine Rabin and Streett condition requiring both big Rabin automata and big Streett automata. Even in this case, the method scales well. Further, Formula 9 contains non-infinitary behaviour, e.g. $\mathbf{G}p_1 \neq 10$. Therefore, the DGRW is of size greater than 1, and thus also the product is bigger as can be seen in the $s_{\mathbf{GR}}/s_i$ column.

4 Synthesis

In this section, we show how generalized Rabin pairs can be used to speed up the computation of a winning strategy in an LTL(**F**,**G**) game and thus to speed up LTL(**F**,**G**) synthesis. A game is defined like an MDP, but with the stochastic vertices replaced by vertices of an adversarial player.

Definition 13. *A* game $\mathcal{M} = (V, E, (V_0, V_1))$ *consists of a finite directed* game graph (V, E) *and a partition* (V_0, V_1) *of the* finite *set* V *of vertices into player-0 vertices* (V_0) *and player-1 vertices* (V_1).

An *LTL game* is a game together with an LTL formula with vertices as atomic propositions. Similarly, a *Rabin game* and a *game with GRP condition (GRP game)* is a game with a set of Rabin pairs, or a set of generalized Rabin pairs, respectively.

A *strategy* is a function $V^* \to E$ assigning to each *history* an outgoing edge of its last vertex. A play conforming to the strategy f of Player 0 is any infinite sequence $v_0 v_1 \cdots$ satisfying $v_{i+1} = f(v_0 \cdots v_i)$ whenever $v_i \in V_0$, and just $(v_i, v_{i+1}) \in E$ otherwise. Player 0 has a *winning strategy*, if there is a strategy f such that all plays conforming to f of Player 0 satisfy the LTL formula, Rabin condition or GRP condition, depending on the type of the game. For further details, we refer to e.g. [PP06].

One way to solve an LTL game is to make a product of the game arena with the DRW corresponding to the LTL formula, yielding a Rabin game. The current fastest solution of Rabin games works in time $\mathcal{O}(mn^{k+1}kk!)$ [PP06], where $n = |V|, m = |E|$ and k is the number of pairs. Since n is doubly exponential and k singly exponential in the size of the formula, this leads to a doubly exponential algorithm. And indeed, the problem of LTL synthesis is 2-EXPTIME-complete [PR89].

Similarly as for model checking of probabilistic systems, we investigate what happens (1) if we replace the translation to Rabin automata by our new translation and (2) if we employ DGRW instead. The latter leads to the problem of GRP games. In order to solve them, we extend the methods to solve Rabin and Streett games of [PP06].

We show that solving a GRP game is faster than first degeneralizing them and then solving the resulting Rabin game. The induced speed-up factor is $|B|^k$. In the following two subsections we show how to solve GRP games and analyze the complexity. The subsequent section reports on experimental results.

4.1 Generalized Rabin ranking

We shall compute a ranking of each vertex, which intuitively states how far from winning we are. The existence of winning strategy is then equivalent to the existence of a ranking where Player 0 can always choose a successor of the current vertex with smaller ranking, i.e. closer to fulfilling the goal.

Let $(V, E, (V_0, V_1))$ be a game, $\{(F_1, \mathcal{I}_1), \ldots, (F_k, \mathcal{I}_k)\}$ a GRP condition with the corresponding degeneralization domain B. Further, let $n := |V|$ and denote the set of permutations over a set S by $S!$.

Definition 14. *A* ranking *is a function $r : V \times B \to R$ where R is the* ranking domain $\{1, \ldots, k\}! \times \{0, \ldots, n\}^{k+1} \cup \{\infty\}$.

The ranking $r(v, wf)$ gives information important in the situation when we are in vertex v and are waiting for a visit of $\mathcal{I}_i^{wf(i)}$ for each i given by $wf \in B$. As time passes the ranking should decrease. To capture this, we define the following functions.

Definition 15. *For a ranking r and given $v \in V$ and $wf \in B$, we define* $\mathrm{next}_v : B \to B$

$$\mathrm{next}_v(wf)(i) = \begin{cases} wf(i) & \text{if } v \notin \mathcal{I}_i^{wf(i)} \\ wf(i) \oplus 1 & \text{if } v \in \mathcal{I}_i^{wf(i)} \end{cases}$$

and $\mathrm{next} : V \times B \to R$

$$\mathrm{next}(v, wf) = \begin{cases} \min_{(v,w)\in E} r(w, \mathrm{next}_v(wf)) & \text{if } v \in V_0 \\ \max_{(v,w)\in E} r(w, \mathrm{next}_v(wf)) & \text{if } v \in V_1 \end{cases}$$

where the order on $(\pi_1 \cdots \pi_k, w_0 w_1 \cdots w_k) \in R$ is given by the lexicographic order $>_{lg}$ on $w_0 \pi_1 w_1 \pi_2 w_2 \cdots \pi_k w_k$ and ∞ being the greatest element.

Intuitively, the ranking $r(v, wf) = (\pi_1 \cdots \pi_k, w_0 w_1 \cdots w_k)$ is intended to bear the following information. The permutation π states the importance of the pairs. The pair $(F_{\pi_1}, \mathcal{I}_{\pi_1})$ is the most important, hence we are not allowed to visit F_{π_1} and we desire to either visit $\mathcal{I}_{\pi_1}$, or not visit F_{π_2} and visit $\mathcal{I}_{\pi_2}$ and so on. If some important F_i is visited it becomes less important. The importance can be freely changed only finitely many (i_0) times. Otherwise, only less important pairs can be permuted if a more important pair makes good progress. Further, w_i measures the worst possible number of steps until visiting $\mathcal{I}_{\pi_i}$. This intended meaning is formalized in the following notion of good rankings.

Definition 16. *A ranking r is* good *if for every $v \in V, wf \in B$ with $r(v, wf) \neq \infty$ we have $r(v, wf) >_{v,wf} \text{next}(v, wf)$.*

We define $(\pi_1 \cdots \pi_k, w_0 w_1 \cdots w_k) >_{v,wf} (\pi'_1 \cdots \pi'_k, w'_0 w'_1 \cdots w'_k)$ if either $w_0 > w'_0$, or $w_0 = w'_0$ with $>^1_{v,wf}$ hold. Recursively, $>^\ell_{v,wf}$ holds if one of the following holds:

- $\pi_\ell > \pi'_\ell$
- $\pi_\ell = \pi'_\ell$, $v \not\models F_{\pi_\ell}$ *and* $w_\ell > w'_\ell$
- $\pi_\ell = \pi'_\ell$, $v \not\models F_{\pi_\ell}$ *and* $v \models \mathcal{I}^{wf(\pi_\ell)}_{\pi_\ell}$
- $\pi_\ell = \pi'_\ell$, $v \not\models F_{\pi_\ell}$ *and* $w_\ell = w'_\ell$ *and* $>^{\ell+1}_{v,wf}$ *holds (where* $>^{k+1}_{v,wf}$ *never holds)*

Moreover, if one of the first three cases holds, we say that $\succ^\ell_{v,wf}$ holds.

Intuitively, $>$ means the second element is closer to the next milestone and $\succ^\ell$, moreover, that it is so because of the first ℓ pairs in the permutation.

Similarly to [PP06], we obtain the following correctness of the construction. Note that for $|B| = 1$, the definitions of the ranking here and the *Rabin ranking* of [PP06] coincide. Further, the extension with $|B| > 1$ bears some similarities with the Streett ranking of [PP06].

Theorem 17. *For every vertex v, Player 0 has a winning strategy from v if and only if there is a good ranking r and $wf \in B$ with $r(v, wf) \neq \infty$.*

4.2 A fixpoint algorithm

In this section, we show how to compute the smallest good ranking and thus solve the GRP game. Consider a lattice of rankings ordered component-wise, i.e. $r_1 >_c r_2$ if for every $v \in V$ and $wf \in B$, we have $r_1(v, wf) >_{lg} r_2(v, wf)$. This induces a complete lattice. The minimal good ranking is then a least fixpoint of the operator Lift on rankings given by:

$$\text{Lift}(r)(v, wf) = \max\left\{r(v, wf), \min\{x \mid x >_{v,wf} \text{next}(v, wf)\}\right\}$$

where the optima are considered w.r.t. $>_{lg}$. Intuitively, if Player 0 cannot choose a successor smaller than the current vertex (or all successors of a Player 1 vertex are greater), the ranking of the current vertex must rise so that it is greater.

Theorem 18. *The smallest good ranking can be computed in time $\mathcal{O}(mn^{k+1}kk! \cdot |B|)$ and space $(nk \cdot |B|)$.*

Proof. The lifting operator can be implemented similarly as in [PP06]. With every change, the affected predecessors to be updated are put in a worklist, thus working in time $\mathcal{O}(k \cdot \text{out-deg}(v))$. Since every element can be lifted at most $|R|$-times, the total time is $\mathcal{O}(\sum_{v \in V} \sum_{wf \in B} k \cdot \text{out-deg}(v) \cdot |R|) = |B|km \cdot n^{k+1}k!$. The space required to store the current ranking is $\mathcal{O}(\sum_{v \in V} \sum_{wf \in B} k) = n \cdot |B| \cdot k$. □

We now compare our solution to the one that would solve the degeneralized Rabin game. The number of vertices of the degeneralized Rabin game is $|B|$ times greater. Hence the time needed is multiplied by a factor $|B|^{k+2}$, instead of $|B|$ in the case of a GRP game. Therefore, our approach speeds up by a factor of $|B|^{k+1}$, while the space requirements are the same in both cases, namely $\mathcal{O}(nk \cdot |B|)$.

Example 19. A conjunction of two fairness constraints of example 8 corresponds to $|B| = 2$ and $k = 4$, hence we save by a factor of $2^4 = 16$. A conjunction of three fairness constraints corresponds to $|B| = 24$ and $k = 8$, hence we accelerate $24^8 \approx 10^{11}$ times.

Further, let us note that the computation can be implemented recursively as in [PP06]. The winning set is $\mu Z.\ \mathfrak{GR}(\mathcal{GR}, \mathbf{tt}, \heartsuit Z)$ where $\mathfrak{GR}(\emptyset, \varphi, W) = W$,

$$\mathfrak{GR}(\mathcal{GR}, \varphi, W) = \bigvee_{i \in [1..k]} \nu Y. \bigwedge_{j \in [1..|\mathcal{I}_i|]} \mu X.\ \mathfrak{GR}\Big(\mathcal{GR} \setminus \{(F_i, \mathcal{I}_i)\}, \varphi \wedge \neg F_i, W \vee (\varphi \wedge \neg F_i \wedge I_i^j \wedge \heartsuit Y) \vee (\varphi \wedge \neg F \wedge \heartsuit X)\Big)$$

$\heartsuit\varphi = \{u \in V_0 \mid \exists (u,v) \in E : v \models \varphi\} \cup \{u \in V_1 \mid \forall (u,v) \in E : v \models \varphi\}$ and μ and ν denote the least and greatest fixpoints, respectively. The formula then provides a succinct description of a symbolic algorithm.

4.3 Experimental Evaluation

Reusing the notation of Section 3.2, we compare the performance of the methods for solving LTL games. We build and solve a Rabin game using

Ł ltl2dstar producing DRW (from LTL formulae),
R Rabinizer producing DRW, and
GR Rabinizer producing DGRW.

We illustrate the methods on three different games and three LTL formulae; see Table 2. The games contain 3 resp. 6 resp. 9 vertices. Similarly to Section 3.2, s_i denotes the number of vertices in the ith arena, $s_{\mathbf{Ł}}, s_{\mathbf{R}}, s_{\mathbf{GR}}$ the number of vertices in the resulting games for the three methods, and $t_{\mathbf{Ł}}, t_{\mathbf{R}}, t_{\mathbf{GR}}$ the respective running times.

Formula 1 allows for a winning strategy and the smallest ranking is relatively small, hence computed quite fast. Formula 2, on the other hand, only allows for larger rankings. Hence the computation takes longer, but also because in **Ł** and **R** cases the automata are larger than for formula 1. While for **Ł** and **R**, the product is usually too big, there is a chance to find small rankings in **GR** fast. While for e.g. $\mathbf{FG}(a \vee \neg b \vee c)$, the automata and games would be the same for all three methods and the solution would only take less than a second, the more complex formulae 1 and 2 show clearly the speed up.

Table 2. Experimental comparison of **Ł**, **R**, and **GR** methods for solving LTL games. Again the sign "−" denotes either crash, out-of-memory, time-out after 30 minutes, or a ratio where one operand is −.

Formula	s_i	$\frac{s_{\mathbf{Ł}}}{s_i}$	$\frac{s_{\mathbf{R}}}{s_i}$	$\frac{s_{\mathbf{GR}}}{s_i}$	$t_{\mathbf{Ł}}$	$t_{\mathbf{R}}$	$t_{\mathbf{GR}}$	$\frac{t_{\mathbf{R}}}{t_{\mathbf{GR}}}$	$\lvert B\rvert$	$\frac{t_{\mathbf{Ł}}}{t_{\mathbf{GR}}}$
$(\mathbf{GF}a \wedge \mathbf{GF}b \wedge \mathbf{GF}c)$	3	22	7.3	4	63.2	1.6	1.1	1.4	9	48.2
$\vee(\mathbf{GF}\neg a \wedge \mathbf{GF}\neg b \wedge \mathbf{GF}\neg c)$	6	21.3	7.3	3.7	878.6	14.1	7.3	2	9	130.3
	9	20.6	7	3.6	−	54.8	31.3	1.8	9	−
$(\mathbf{GF}a \vee \mathbf{FG}b) \wedge (\mathbf{GF}c \vee \mathbf{GF}\neg a)$	3	21	10	4	−	117.5	12	9.8	6	−
$\wedge(\mathbf{GF}c \vee \mathbf{GF}\neg b)$	6	16.2	9.2	3.7	−	−	196.7	−	6	−
	9	17.6	9.2	3.6	−	−	1017.8	−	6	−

5 Conclusions

In this work we considered the translation of the LTL(**F**,**G**) fragment to deterministic ω-automata that is necessary for probabilistic model checking as well as synthesis. The direct translation to deterministic Muller automata gives a compact automata but the explicit representation of the Muller condition is huge and not algorithmically amenable. In contrast to the traditional approach of translation to deterministic Rabin automata that admits efficient algorithms but incurs a blow-up in translation, we consider deterministic automata with generalized Rabin pairs (DGRW). The translation to DGRW produces the same compact automata as for Muller conditions. We presented efficient algorithms for probabilistic model checking and game solving with DGRW conditions which shows that the blow-up of translation to Rabin automata is unnecessary. Our results establish that DGRW conditions provide the convenient formalism that allows both for compact automata as well as efficient algorithms. We have implemented our approach in PRISM, and experimental results show a huge improvement over the existing methods. Two interesting directions of future works are (1) extend our approach to LTL with the **U**(until) and the **X**(next) operators; and (2) consider symbolic computation and Long's acceleration of fixpoint computation (on the recursive algorithm), instead of the ranking function based algorithm for games, and compare the efficiency of both the approaches.

References

AT04. R. Alur and S. La Torre. Deterministic generators and games for LTL fragments. *ACM Trans. Comput. Log.*, 5(1):1–25, 2004.

BK08. C. Baier and J.-P. Katoen. *Principles of model checking.* MIT Press, 2008.

Chu62. A. Church. Logic, arithmetic, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35. Institut Mittag-Leffler, 1962.

Ehl11. R. Ehlers. Generalized rabin(1) synthesis with applications to robust system synthesis. In *NASA Formal Methods*, pages 101–115, 2011.

GKE12. A. Gaiser, J. Křetínský, and J. Esparza. Rabinizer: Small deterministic automata for ltl(f, g). In *ATVA*, pages 72–76, 2012.

JB06. B. Jobstmann and R. Bloem. Optimizations for LTL synthesis. In *FMCAD*, pages 117–124. IEEE Computer Society, 2006.

JGWB07. B. Jobstmann, S. J. Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In *CAV*, volume 4590 of *LNCS*, pages 258–262. Springer, 2007.

KB06. J. Klein and C. Baier. Experiments with deterministic ω-automata for formulas of linear temporal logic. *Theor. Comput. Sci.*, 363(2):182–195, 2006.

KB07. J. Klein and C. Baier. On-the-fly stuttering in the construction of deterministic ω-automata. In *CIAA*, volume 4783 of *LNCS*, pages 51–61. Springer, 2007.

KE12. J. Křetínský and J. Esparza. Deterministic automata for the (f, g)-fragment of ltl. In P. Madhusudan and Sanjit A. Seshia, editors, *CAV*, volume 7358 of *Lecture Notes in Computer Science*, pages 7–22. Springer, 2012.

Kle. J. Klein. ltl2dstar - LTL to deterministic Streett and Rabin automata. http://www.ltl2dstar.de/.

KNP11. M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

Kup12. O. Kupferman. Recent challenges and ideas in temporal synthesis. In *SOFSEM*, volume 7147 of *LNCS*, pages 88–98. Springer, 2012.

KV05. O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *FOCS*, pages 531–542. IEEE Computer Society, 2005.

MS08. A. Morgenstern and K. Schneider. From LTL to symbolically represented deterministic automata. In *VMCAI*, volume 4905 of *LNCS*, pages 279–293. Springer, 2008.

Pit06. N. Piterman. From nondeterministic Buchi and Streett automata to deterministic parity automata. In *LICS*, pages 255–264, 2006.

Pnu77. A. Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

PP06. N. Piterman and A. Pnueli. Faster solutions of rabin and streett games. In *LICS*, pages 275–284, 2006.

PPS06. N. Piterman, A. Pnueli, and Y. Sa'ar. Synthesis of reactive(1) designs. In *VMCAI*, volume 3855 of *LNCS*, pages 364–380. Springer, 2006.

PR89. A. Pnueli and R. Rosner. On the synthesis of an asynchronous reactive module. In *ICALP*, volume 372 of *LNCS*, pages 652–671. Springer, 1989.

Put94. M.L. Puterman. *Markov Decision Processes*. 1994.

PZ86. A. Pnueli and L. Zuck. Verification of multiprocess probabilistic protocols. *Distributed Computing*, 1(1):53–72, 1986.

Saf88. S. Safra. On the complexity of ω-automata. In *FOCS*, pages 319–327. IEEE Computer Society, 1988.

VW86. M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):183–221, 1986.

A Proof of Theorem 17 (correctness of the ranking)

A.1 Soundness

Lemma 20. *For every good ranking r and vertex v with $r(v, wf) \neq \infty$ for some $wf \in B$, Player 0 has a winning strategy from v.*

Proof. We construct a strategy with memory B and memory update $(v, wf) \mapsto \mathrm{next}_v(wf)$. When in vertex v with memory wf, the strategy chooses a successor v' for which $r(v', \mathrm{next}_v(wf)) = \mathrm{next}(v, wf)$, i.e. with the lowest admissible ranking. We prove it is winning from v.

Consider an infinite play $v^0v^1\cdots$ conforming to the strategy and $wf^0wf^1\cdots$ the corresponding memories and $r^0r^1\cdots$ the corresponding ranks $r^i = r(v^i, wf^i) = (\pi^i_1 \cdots \pi^i_k, w^i_0 \cdots w^i_k)$. By definitions 16 and 15, $r^i >_{v^i, wf^i} r^{i+1}$ for all i.

Let ℓ be the smallest number for which $r^m \succ^{\ell}_{v^m, wf^m} r^{m+1}$ for infinitely many m. Then for almost all i

- r^i are the same on the first ℓ elements of both components, i.e. on each of π^i_1 to π^i_ℓ and w^i_0 to $w^i_{\ell-1}$, we denote the repetitive π^i_ℓ by *win*,
- thus also $v^i \not\models F_{win}$, and
- wf^i are the same on the first $\ell - 1$ elements,

thus since $[0..n]$ is well founded, $wf^j(win)$ gets all values from $[1..|\mathcal{I}_{win}|]$ infinitely often and $v^j \models \mathcal{I}^k_{win}$ infinitely often for each k and the winth pair is satisfied. □

A.2 Completeness

We use the standard μ-calculus and define an operator $\heartsuit$ of the *controllable predecessor* as follows:

$$\heartsuit\varphi = \{u \in V_0 \mid \exists (u, v) \in E : v \models \varphi\} \cup \{u \in V_1 \mid \forall (u, v) \in E : v \models \varphi\}$$

Further, we define recursively

$$\mathfrak{GR}(\emptyset, \varphi, W) = W$$

$$\mathfrak{GR}(\mathcal{GR}, \varphi, W) = \bigvee_{i\in[1..k]} \nu Y. \bigwedge_{j\in[1..|\mathcal{I}_i|]} \mu X.\, \mathfrak{GR}\Big(\mathcal{GR} \setminus \{(F_i, \mathcal{I}_i)\}, \varphi \wedge \neg F_i, W \vee (\varphi \wedge \neg F_i \wedge I^j_i \wedge \heartsuit Y) \vee (\varphi \wedge \neg F \wedge \heartsuit X)\Big)$$

Let $\mathfrak{Win}(\mathcal{GR})$ be the set of winning vertices of Player 0 with the winning condition being the set $\mathcal{GR}$ of GRPs. Then a simple adaptation of Claim 9 of [PP06] to the setting with the conjunction yields an alternative characterization of the winning set.

Lemma 21. *For non-empty $\mathcal{GR}$, the set $\mathfrak{GR}(\mathcal{GR}, \varphi, W)$ is the winning region for Player 0 and the winning condition*

$$\bigvee_{(F,\mathcal{I})\in\mathcal{GR}} \left((\varphi \wedge \neg F)\mathbf{U}W \vee \mathbf{G}\Big(\varphi \wedge \neg F \wedge \bigwedge_{j\in[1..|\mathcal{I}|]} \mathbf{F}\mathcal{I}^j\Big) \vee \Big(\mathfrak{Win}(\mathcal{GR} \setminus \{(F, \mathcal{I})\}) \wedge \mathbf{G}(\varphi \wedge \neg F)\Big)\right)$$

As a result, we get directly by [PP06] the following symbolic recursive algorithm.

Lemma 22. $\mathfrak{Win}(\mathcal{GR}) = \mu Z.\ \mathfrak{GR}(\mathcal{GR}, \mathbf{tt}, \heartsuit Z)$

It remains to show a good ranking on the vertices of $\mathfrak{Win}(\mathcal{GR})$ using the characterization above.

Lemma 23. *There is a good ranking such that for every vertex v, from which Player 0 has a winning strategy, there is a permutation $wf \in B$ with $r(v, wf) \neq \infty$.*

Proof. We show how to define good ranking on $\mathfrak{Win}(\mathcal{GR})$. To this end, we use the characterization above written more explicitly in the following algorithm (where all variables are local):

```
Function mainGR(SetOfPairs)
    LeastFix(Z)
        Z := Rabin(SetOfPairs, true, ♡Z)
    End – LeastFix(Z)
    Return Z
End

Function GR(SetOfPairs, Invariant, AlreadyOk)
    Win := 0
    Foreach (F, I) ∈ SetOfPairs
        RemainingPairs := SetOfPairs\{(F, I)}
        GreatestFix(Y)
            Foreach j ∈ {1..|I|}
                conjunctionX := true
                LeastFix(X)
                    NewOk := AlreadyOk ∪ (Invariant ∩ ¬F ∩ I^j ∩ ♡Y) ∪ (Invariant ∩ ¬F ∩ ♡X)
                    If (|RemainingPairs| = 0)
                            X := NewOk
                    Else
                            X := Rabin(RemainingPairs, Invariant ∩ ¬F, NewOk)
                    End – If (|RemainingPairs| = 0)
                End – LeastFix(X)
                Let conjunctionX := conjunctionX ∩ X
            End – Foreach j
            Let Y := conjunctionX
        End – GreatestFix(Y)
        Let Win := Win ∪ Y
    End – Foreach (F, I)
    Return Win
End
```

Similarly to [PP06], we monitor the call stack of the procedure. We assgin a counter i to each least fixpoint (Z and all nested X's), starting from zero increasing every time next iteration is done. We consider configurations of the program where all greatest fixpoints are in their last iteration.

Each of the states returned by mainGR gets a rank according to the first time it is discovered by the least fixpoints. For a given configuration, let $p_1 \cdots p_k$ be the pairs handled by the nested calls of GR and $i_1 \cdots i_k$ the k nested values of i's, i.e. the numbers of iterations of the nested least fixpoints (considering the last calls of the greatest fixpoints) and i_0 the value of the counter for Z, and $j_1 \dots j_k$ the nested values of j. Let $X_{p_1 \cdots p_k}^{i_0 i_1 \cdots i_k}$ be the current values of the intersection X.

Now for every vertex v in the returned set, and $wf \in B$, we set $r(v, wf)$ to be the smallest (w.r.t. $>_{lg}$) element $(p_1 \cdots p_k, i_0 i_1 \cdots i_k)$ of the ranking domain where $v \in X_{p_1 \cdots p_k}^{i_0 i_1 \cdots i_k}$ and $wf(\pi_n) = j_n$ for all n.

All other pairs (v, wf) get rank ∞.

The ranking can now be easily shown to be good following [PP06], since in order to discover a state, its successors must have been already discovered before and have thus a smaller ranking. □

B Figures

We give figures of the game family used for the experimental evaluation in section 4.3.

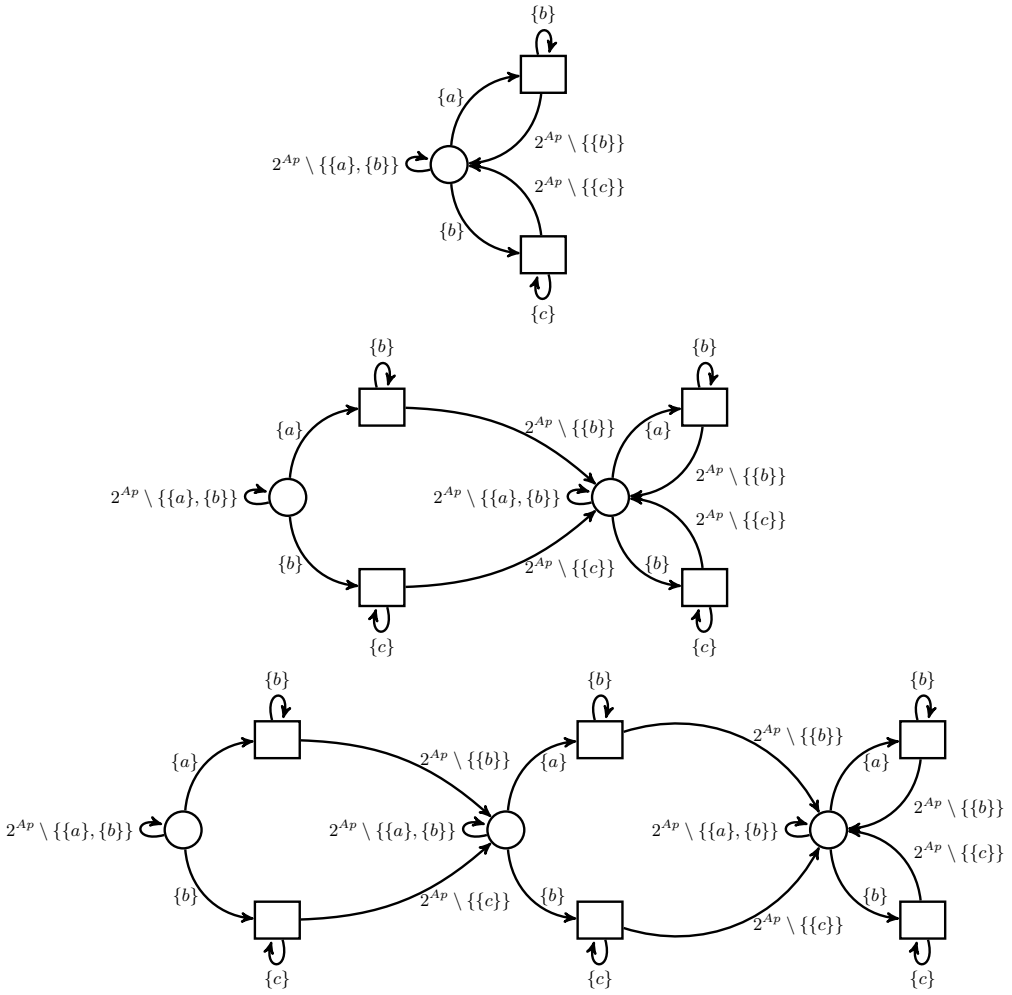


Fig. 1. Games used for the experiments in section 4.3, with $Ap = \{a, b, c\}$.